SAMPLE SUBMISSION

Title

Battle of the SKM and IUM: How Windows 10 Rewrites OS Architecture

Alex Ionescu, Chief Architect, CrowdStrike
(Black Hat USA 2015)

Track

OS - Host and Container Security

Abstract

Notes:

- *Detailed, yet concise abstract*
- *Defines a problem and offers a solution(s) that will be examined during session*

In Windows 10, Microsoft is introducing a radical new concept to the underlying OS architecture, and likely the biggest change to the NT design since the decision to move the GUI in kernel-mode.

In this new model, the Viridian Hypervisor Kernel now becomes a core part of the operating system and implements Virtual Secure Machines (VSMs) by loading a true microkernel - a compact (200kb) NT look-alike with its own drivers called the Secure Kernel Mode (SKM) environment, which then uses the Hypervisor to hook and intercept execution of the true NT kernel. This creates a new paradigm where the NT Kernel, executing in Ring 0, now runs below the Secure Kernel, at Ring ~0 (called Virtual Trust Level 1).

But it doesn't stop there - as the Ring 0 NT kernel now has the ability to not only create standard Ring 3 user-mode applications, but also Ring ~3 applications (or Virtual Trust Level 0) that run in Isolated User Mode (IUM). Because VTLs are all more privileged than Ring 0, this now creates a model where a user-mode application running inside a VSM now has data and rights that even the kernel itself cannot modify.

Why go through all this trouble? Because it seems like the hottest thing these days is Pass-the-Hash, and attacks must seemingly be mitigated at all costs. And even in Windows 8.1, an attacker with the permissions to load a kernel driver can bypass the existing mitigations (and Mimikatz is signed!). With VTLs, now even the most privileged attacker is only as privileged as the hypervisor will allow it - never able to truly read the hash date that is stored in the secure partition.

How "secure" is this new model really? And what prevents a malicious application from running in such a secure mode to begin with?

Presentation Outline

Notes:

- *Clearly conveys progression of talk*
- *Helps the Review Board visualize the presentation in its entirety*
- *Gives an explanation of each area of the presentation*

1. Introduction to key terms (VTL, VSM, SKM, IUM, etc...)
- This will detail the basic terminology

2. SKM Boot Architecture and Hypervisor Support
- This will detail how the secure kernel is started up, how to configure a system for SKM support, and the key new Hyper-V 4.0 hypercalls that are used to initialize SKM. We will then go over the SKM boot process and its initialization, including discussion of the IDK (Identification Key) and LK (Local Key)

3. SKM to Insecure Kernel Communication
-Once SKM is initialized, this will discuss how it interacts with the vanilla NT kernel and the interfaces that are created in between, managed by the hypervisor. We will describe all the SKM calls that exist from SKM to NT kernel.

4. IUM Initialization
-Next, this will go over how the Isolated User Mode environment starts itself up, and how to create a "secure process" or a "trustlet". We'll also talk about Vmsp.exe (Virtual Machine Secure Process) worker host.

5. IUM to Insecure Kernel Communication
-Here, we'll detail how a trustlet performs system calls and how its environment is sandboxed and allows communication to the real vanilla kernel.

6. IUM to Secure Kernel Communication
-Continuing on the previous topic, this will explain how a Trustlet actually talks to SKM, and how it can obtain the key secure data that the SKM is protecting. We will also go over all the Secure System Calls that are implemented by the SKM.

7. Live Demo of IUM "Trustlet"

-We will take a look at LSASS, implemented as a Trustlet, as well as a custom Trustlet I've written to demonstrate some of the protections afforded and how the isolation works. We will also look at how a potentially malicious Trustlet could attack the system.

8. Closing Remarks
-Finally, this section will show potential avenues for abuse of the system, as well as some thoughts on the practicality of the implementation of this system in Windows 10.

Attendee Takeaways

Notes:

- *Submitter fulfilled requirement of providing 3 takeaways*
- *Explains relevance to the audience*
- *Cleary emphasizes the participant benefits*

1. First, that the security model in Windows 10 is radically changing -- obtaining root/kernel privileges on the machine only gives you Ring 0 rights to the virtualized Hyper-V partition, and not the entire system. There is a new, extra, security boundary that must be broken for full access.

2. Second, how this relates to PtH attacks today, and how future attacks might look like.

3. Third, how this can be enabled and what its requirements are - as well as its weaknesses.

And finally, a good dose of Windows Internals, as always :)

Why Black Hat?

Notes:

- *Summarizes the scale of the issue and its potential impact*
- *Argues relevance/importance of the presentation at Black Hat*

This is a potentially huge change to the most widely used Operating System with far-reaching consequences to OS security, virtualization, safe cryptographic storage, and more. The idea that even the host OS itself is now virtualized becoming mainstream, having started from the first "Blue Pill" esoteric attack to finally being the norm in Windows 10. Black Hat is the perfect platform to make everyone aware of this drastic new model.

SAMPLE SUBMISSION

Title

Notes:

Eye-catching but relevant title

Big Game Hunting: The Peculiarities of Nation-State Malware Research

- Morgan Marquis-Boire, Security Engineer, Citizen Lab, University of Toronto
- Marion Marschalek, Principal Malware Researcher, G DATA Advanced Analytics
- Claudio Guarnieri
(Black Hat USA 2015)

Track

Malware Defense

Abstract

Notes:

- *Provides a detailed introduction and summary of the content to be covered*
- *Defines a problem and offers a solution(s) that will be examined during session*

The security industry focus on state-sponsored espionage is a relatively recent phenomenon. Since the Aurora Incident brought nation-state hacking into the spotlight, there's been high profile reports on targeted hacking by China, Russia, U.S.A, Israel, to name a few. This has lead to the rise of a lucrative Threat Intelligence business, propelling marketing and media campaigns and fueling political debate.

This talk will cover the idiosyncrasies of nation-state malware research using the experiences of presenters in the 'Threat Analyst Sweatshop'. Regin (aka WARRIORPRIDE, allegedly written by the Five Eyes) and Babar (aka SNOWGLOBE, allegedly written by France) will be used as case studies in examining attribution difficulties. Additionally, well examine attributing commercially written offensive software (implants and exploits) and the (mostly negative) vendor responses. Well cover what happens when you find other players on the hunt, and address the public misconception that attribution is frequently done using open source information.

We will focus on the attribution problem and present a novel approach on creating credible links between binaries originating from the same group of authors. Our goal is to add to transparency in attribution and supply analysts with a tool to emphasize or deny vendor statements. The technique is based on features derived from different domains, such as implementation details, applied evasion techniques, classical malware traits or infrastructure attributes; which are then leveraged to compare the handwriting among binaries.

Presentation Outline

Notes:

- *Thoroughly conveys the framework of the presentation*
- *Helps the Review Board visualize the presentation in its entirety*
- *Gives an explanation of each area of the presentation, rather than simply listing the steps*
- *(Please note – the below length is not required, though we do request a thoughtfully-prepared structure)*

Problem Statement

In malware research, especially when dealing with APT and nation state campaigns, we face the challenge of putting the malware at hand into context. Vendors and nation-states are obliged to deny attribution, even if correct. Furthermore, a binary itself does not give away who wrote it, who controlled it, who the infected victims were or what the aim of the operation that involved it was. A standalone binary does not even tell which operation involved it.

We cannot conclude from a binary to its context. What we can do though, is posit from a binary to a related binary, which in most cases helps a great deal in the investigation. Bringing a malicious binary in context with known APT malware often allows credible conclusions about the nature of an attack or in a few limited cases even allows linking a binary with an alleged operator.

In malware research conclusions are very often based on non-scientific research. We will present the audience with an approach that helps building credible links between a binary and a given set of binaries from the same author in a measurable way.

Related Research

A number of research projects focus on authorship attribution of code, commonly named code stylometry. The aim of these projects is to identify the origin of a piece of source code or a binary within a set of repositories by known authors.

Code stylometry most certainly does not work on binaries as compilers and disassemblers (or decompilers) wash away most attributes mentioned in recent publications. Caliskan [1] describes attributes, for example, the average number of characters per word, character count, use of special characters, punctuation and percentage of digits as feasible for source code attribution. These cannot be applied to disassembled or decompiled binaries.

Burrows, Uitdenbogerd and Turpin [3] base their authorship attribution research on a feature set including white spaces, operators, literals, keywords, I/O words and function words from standard C libraries. Again, considering the loss of information through disassembly or decompilation this approach will most certainly fail.

Research published by Rosenblum, Zhu and Miller [4] describes methodologies to automatically detect stylistic features of binary code. Their approach derives features from a control flow graph representation, the instruction sequence of a binary to build a classifier based on support vector machines, as well as the clustering of program authors based on the k-means algorithm. The set of derived features are n-grams and idioms from an instruction level; graphlets, supergraphlets and call graphlets from the control flow graph as well as library calls.

While this is probably the closest to putting two binaries in a context, the approach comes with weaknesses. Regarding APT- or nation state malware, one or more families of malicious binaries do not stem from one author but most likely from a team of developers with switching team members, who have been working on the malware over a long period of time. Additionally, the described technique does not consider standard library code within a binary, deal with obfuscated binaries or show resistance against attribution evasion or false flag techniques. At the same time numerous aspects of malware are not considered for classification, especially with regard to APT and nation state malware.

The Approach

We will present a novel approach on creating credible links between binaries originating from the same group of authors. The technique is based on features derived from different domains, such as implementation details, applied evasion techniques, classical malware traits or infrastructure attributes. Such features could for example be shared memory allocation habits, obfuscation or custom encryption algorithms, reused anti-simulation tricks, shared system infiltration- or persistence techniques or shared command and control servers.

Deriving features from various domains helps avoid sole reliance on the binary representation of a sample. On the instruction level a program can be altered with something as simple as a change in compiler settings. Also, a broader set of attributes counters deliberate authorship confusion techniques. It is assumed that attributable features are possibly faked, but we rely on the fact that it is difficult to fake all features on all levels. The question finally is, how many features can we gather from compiled binaries and how many do we need to create a credible link?

We will introduce a catalog of attributes which have proven to be feasible in identifying related binaries and discuss options for automated extraction. We will also discuss the information gained from the dedicated features and show their applicability. Our interest lies in features that are more likely to be introduced by the program author than from outside, from the compiler or linker for example.

The following list is a non-exhaustive draft of the catalog to be presented:

- String constants
- Error messages
- String formatting style
- English grammar mistakes
- C&C commands
- Timestamp formatting
- Implementation traits
- Memory allocation habits
- Use of global variables
- Multi-threading model
- Software architecture and design
- Constructor design
- Dynamic API loading technique
- Exception handling
- Usage of public source code from same sources
- Programming language and compiler
- Compilation time stamps and time zones
- Custom features
- Obfuscation techniques
- Stealth and evasion techniques
- Use of encryption and compression algorithms
- Shared encryption keys
- Re-used source code in general
- Malware specific features
- System infiltration
- Propagation mechanisms
- Artifact naming schemes/algorithms
- Data exfiltration techniques
- System/OS version determination technique
- C&C command parsing implementation
- Infrastructure
- Shared C&C servers
- Overlapping countries/languages used for domain hosting and naming
- Shared beaconing style
- Communication protocol/port

Communication Intervals

In order to maintain the practical use of this approach the derived features need to be normalized and stored in an abstract form. Feature extraction is an overall tedious process and for most features has to be performed by an analyst by hand. Once the data is complete and abstracted though it is easy to compare and extend as needed.

Case Study on FinFisher/Hacking Team/VUPEN
Our publications on the FinFisher and RCS commercial spyware opened the field to investigate the systematic use of hacking by Western law enforcement as well as regimes around the world.

Despite numerous reports with reproducible scientific results, spyware vendors like FinFisher and HackingTeam have been consistently denied these discoveries and the weight of the uncovered abuses. Leaked internal documents, however, showed the uncontestable nature of the attribution. Binary attributes, certificates and network behavior have provided systematic ways to monitor the use of such products for an extended period of time.

Case Study on Regin Malware
Regin has been one of the most peculiar cases among the trove of nation-state attacks that have come to light in the last few years. With malware samples spanning over a timeframe of more than 10 years, Regin is one of the largest and most sophisticated malware frameworks discovered to date.

Technical analysis by the authors tied with corroborating evidence provided by leaked documents made Regin (aka WARRIORPRIDE) to be one of the first malware kits to be conclusively attributed to a Western nation-state actor[5]. Consequently, it also represented one of the first instances where security vendors had to face their political biases and publicly justify several years of silence on hacking by governments which are their customers.

Case Study on Animal Farm
The recently uncovered Animal Farm malware and its associated families serve as a useful example for our proposed method, as the binaries are straight forward and easily understood. They come with many of the described features. Additionaly, a fairly large set of binaries have been identified, estimated to be developed over a span of several years and by different authors, while showing very different functionality.

The different families of Animal Farm are NBOT, TFC, Bunny, Babar, Casper and Dino; respectively DDoS malware, reconnaissance malware, espionage tools and scripting bots. Their intentions are very different, yet the list of shared features applied through the introduced catalog is long.

By extracting the before mentioned data for the different families we can prove without doubt that NBOT, TFC, Bunny, Babar, Casper and Dino originate from the same group of authors. Furthermore, we will show how one can apply this system to known APT malware to evaluate their relations.

Sources

[1] http://www.ieee-security.org/TC/SP2013/posters/Aylin_Caliskan_Islam.pdf
[2] https://www.cs.drexel.edu/~ac993/papers/Aylin_Oakland_2013_poster.pdf
[3]http://www.researchgate.net/profile/Alexandra_Uitdenbogerd/publication/220787332_Application_of_Information_Retrieval_Techniques_for_Source_Code_Authorship_Attribution/links/0912f51181dd371820000000.pdf
[4] ftp://ftp.cs.wisc.edu/paradyn/papers/Rosenblum11Authorship.pdf
[5] https://firstlook.org/theintercept/2014/11/24/secret-regin-malware-belgacom-nsa-gchq/
[6] http://motherboard.vice.com/read/meet-babar-a-new-malware-almost-certainly-created-by-france

---

Attendee Takeaways

Notes:

- *Submitter fulfilled requirement of providing 3 takeaways*
- *Highlights the impact on audience*
- *Emphasizes the participant benefits*

(1) Advances the state of public knowledge in the field of nation-state malware research.

(2) In depth analysis into state-of-the-art current events.

(3) We'll reveal something new and awesome!

Why Black Hat?

Notes:

- *Highlights the use of real-world case studies*
- *Argues the impact of the subject matter's relevance at Black Hat*

The presented approach enables analysts to credibly link related binaries and prove or deny a relation among families of binaries.

As experienced by real-world case studies, by being able to prove the different families stem from the same group of authors, one can conclude the various interests of the malware operator by considering the capabilities of the malware. Also it is interesting to derive a higher level of information, such as increasing coding capabilities of the authors or added code features over time. Furthermore, the technique can help prove that the same malware is being used in different operations, thus show that one actor has expanded interests or the same malware strain is being used by different actors.

On the other hand, in reality it is often even more useful to prove the opposite, namely that two malware strains do not carry similar handwriting.